基于 Spark 的并行 Eclat 算法 *

冯兴杰 [a, b]，潘　轩 [a†]

(中国民航大学 a. 计算机科学与技术学院; b. 信息网络中心, 天津 300300)

摘　要：通过对 Spark 大数据平台以及 Eclat 算法的深入分析，提出了基于 Spark 的 Eclat 算法(即 SPEclat)。针对串行算法在处理大规模数据时出现的不足，该方法在多方面进行改进：为减少候选项集支持度计数带来的损耗，改变了数据的存储方式；将数据按前缀进行分组，并划分到不同的计算节点，压缩数据的搜索空间，实现并行化计算。最终将算法结合 Spark 云计算平台的优势加以实现。实验表明该算法可在处理海量数据集时高效运行，并且在面对数据量大规模增长的情况下，具备良好的可扩展性。

关键词：关联规则挖掘；大数据；Spark；投影树；并行化

中图分类号：TP301.6　　doi: 10.3969/j.issn.1001-3695.2017.07.0695

Eclat algorithm based on Spark

Feng Xingjie[a, b], Pan Xuan[a†]

(*a. College of Computer Science & Technology*, *b. Information Network Center*, *Civil Aviation University of China*, *Tianjin* 300300, *China*)

Abstract: After analyzing platform-Spark and the projection tree algorithm, propose the Projection tree algorithm based on Spark (PTBS, Projection Tree algorithm based on Spark) . In view of the shortcomings of the serial algorithm in dealing with large-scale data, the method has been improved in many aspects: To reduce the cost caused by the support count of candidate itemsets, change the data storage structure; data is grouped by prefix and divided into different computing nodes to compress the searching space of data and realize parallel computation. Finally implemente the algorithm on the Spark platform. Experiments show that the algorithm can run efficiently when dealing with massive data, and has good expansibility in the face of large scale data growth.

Key Words: association rules mining ; big data ; Spark ; tree projection ; parallelization

0　引言

在数据挖掘中，频繁模式挖掘[1] 是重要的研究方向之一，过去人们提出了一系列算法来进行频繁模式挖掘，如 Apriori[2]、FP-Growth[3]和 Eclat[4]。互联网技术的飞速发展加速了大数据信息化时代的来临，随之而来的是数据量的增长和计算难度的增加，传统数据挖掘方式已经难以为继。如今，针对大规模数据集，为了产生具有一定价值的数据信息，算法需对原始数据进行反复迭代式地运算，这使得数据挖掘变得异常繁琐。分布式和并行计算方法可以有效地解决对大数据量的存储和运算的问题，同时它还兼备资源共享、高透明性、高性价比、高可靠性、高度灵活性等特点，是处理大数据集的最佳策略。

基于 Hadoop 环境的 MapReduce[5]和 Spark 云计算平台是最为常用的分布式编程运算框架。近年来许多学者针对 MapReduce 框架对传统算法进行了并行化改造。与 MapReduce 框架相比，Spark 编程框架是 MapReduce 思想的一种更加精致和高级的实现，对其过于死板，I/O 密集，不利于构造迭代式算法以及执行效率不高等问题作出了重大改进。在 Eclat 算法并行化方面，文献[9]提出基于 Map /Reduce 的 Eclat 并行算法 PEclat(parallel Eclat)算法。该算法需要多次迭代运行 Map /Reduce 任务，导致性能较低；文献[10]在 Map/Reduce 框架下提出了一种基于前缀分组的并行挖掘算法 MREclat，该算法在对候选 k 项集进行计数时，利用生成候选项集的两个频繁 $(k-1)$ 项集对应的事务序列求交，来计算候选项集的支持度，这样会产生较大的运算量，且随着数据量的增大，运算量呈指数形式增长，影响了算法的实用性；Qiu H 等人则首次在 Spark 上实现了 Apriori 算法的并行化[12] 并提出 YAFIM 算法，该算法采用水平数据结构实现频繁项集的挖掘，并解决了串行算法在 Spark 框架实现并行过程中遇到的诸多问题，但是在得到频繁项集的过程中，需多次遍历原始数据集，将大量的时间浪费在

基金项目：国家自然科学基金委员会与中国民用航空局联合基金项目（U1233113）；国家自然科学青年基金资助项目（61301245，61201414）

作者简介：冯兴杰（1969-），男，河北邢台人，教授，硕导，主要研究方向为数据库及数据仓库、智能信息处理理论与技术；潘轩（1992-），男，山西省晋中市人，硕士研究生，主要研究方向为数据挖掘，大数据技术（465686838@qq.com）.

重复遍历不必要的数据上，致使算法的运行效率不太理想。

本文提出了一种基于 Spark 模型的 Eclat 算法(即 SPEclat)，基本思路是: a)将原数据集的水平结构变换为垂直结构，并用 BitSet 表示项集对应的事物序列，以期实现对频繁项集生成方式的改进，提升运算效率；b)将数据按前缀进行分组，并划分到不同的计算节点，实现算法并行化以充分利用分布式计算环境多节点、高性能的特点。实验结果表明，SPEclat 算法有着较好的可扩展性和加速比，而且比文献[10]提出的 MREclat 算法效率更高。

1　相关技术及方法

1.1　Spark 并行计算框架

Spark 是当前大数据领域最活跃的并行编程模型开源项目之一，Spark 云计算平台因其具有可通过基于内存的方式快速计算，并具备 RDD 存储的持久化和高容错性等特点，成为最适宜于处理大规模数据集的迭代式运算工具。

Spark 能够实现基于内存计算的基础，是其可实现将内存数据抽象为弹性分布式数据集（RDD）。RDD 是一种基于分布式内存的并行数据结构，它能将用户数据存储在内存当中，并控制分区划分从而达到数据分布的目的，同时还具备了数据流的特点：自动容错、位置感知调度和可伸缩性。Spark 还为 RDD 提供了丰富的内置操作，不仅实现了常规的 map 函数和 reduce 函数，还提供了更为丰富的算子，如 foreach、groupBy、join 等，这些算子可以将一个 RDD 通过某一个特定的操作转换为另一个 RDD，这是 Spark 基于内存计算的基本方式。然而，Spark 最重要的功能，是 RDD 的持久化(persistence)操作。数据持久化在内存当中，对于需要多次迭代使用的数据，省去了多次载入到内存或存储到磁盘的过程，极大地加快了数据处理的速度。因此 Spark 编程框架特别适宜于迭代式频繁项集挖掘算法的实现，并行化实现之后，算法的性能会得到非常大的提高。

1.2　频繁模式挖掘算法

1）Apriori 算法：

Apriori[2]使用一种逐层完备搜索的迭代式算法，该搜索算法用到了项集的反单调性，即以“如果一个项集是非频繁的，那么它的所有超集也是非频繁的”为基础，采用连接，剪枝的方式产生候选项集，以压缩候选项集的数量。该算法会对原始数据进行多次遍历；而且在原始数据集中，一些多余事务被用于反复地遍历搜索，而 Apriori 算法并未能将其删减，致使 I/O 消耗增大，降低了算法的运算效率。

2）Eclat 算法

Eclat[4]算法是一种基于垂直数据结构（项的名称 item；包含该项的事物的标志符的集合 Tidset）的频繁项集挖掘算法，以垂直结构数据表示可以避免重复遍历原数据集。项集的支持度计数即为 TidSet 的长度。该算法以概念格理论为基础，采用前缀等价关系划分搜索空间，由具备一定条件的两个项集的并产生候选项集，再通过对应 TidSet 交叉计数来计算支持度，得出频繁项集。在实际应用中有较好的效果，是一种性能较好的频繁项集挖掘算法，算法的伪代码如下所示

Eclat 算法

F_k ={ $I_1,I_2,I_3,L\ ,I_n$ }
Eclat（ F_k ）{
　for all $I_i \in F_k$
　　$F_k = \varnothing$；
　　for all $I_j \in F_k$ ，i< j
　　　N = $I_i\ I\ I_j$
　　　if N.sup ≥ min_sup then
　　　　$F_{k+1} = F_{k+1}\ U\ N$
　　　end
　　end
　end
　if $F_{k+1} = \varnothing$ then
　　Eclat（ F_{k+1} ）
　end
}

该算法自底向上的搜索方式与 Apriori 算法相似，但因数据存储结构的不同，Eclat 算法无须重复遍历数据集。然而，在求候选项集支持度时，如果 TidSet 规模庞大，TidSet 的交集操作会消耗大量时间，影响算法效率。本文对算法作出如下改进：1)用 BitSet 表示事务序列，将支持度的计算简化为对应 BitSet 求与；2)将数据以前缀进行分组，并划分到不同的计算节点，以达到并行化计算的目的。

2　基于 Spark 的 Eclat 算法—SPEclat

本文提出的 SPEclat 算法是一种基于 Spark 的 Eclat 实现，该算法分为三个步骤：a)将数据从 HDFS 中读入并修改数据的存储结构，过滤得到频繁 1 项集，再把结果以 RDD 的形式存储在集群的各个节点的内存当中;b)迭代式地对频繁项集按照相同前缀进行划分;c)在各个计算节点中，以自底向上的方式，并行地由频繁 k 项集得到频繁（k+1）项集，直到没有频繁项集产生。本文第一阶段利用 BitSet 存储事务序列，并以 BitSet 求与的方式代替事务集合求交的运算，加快了频繁项集的生成速率;第二阶段则将获得的数据按前缀分发于不同的计算节点;第三阶段使用改进的 Eclat 算法在各计算节点上求解其所对应的分组中的所有频繁项目集

2.1　第一阶段

首先，直接把存储于 HDFS 中的数据以 Spark RDD 的形式读取并保存，map 任务会将数据读入，并以行的形式对数据进行处理。其中，每一行事务由 Tid 及该事物所包含的所有项组成。

Eclat 算法是以垂直结构存储和处理数据的。所以需要在数据的组织方式上进行预处理。原数据以(Tid,Items)的形式保存于 RDD 中(Items 表示该事务包含的项)。首先在 map 阶段将每

个事务中包含的所有项及对应事务编号形成键值对，然后在 reduce 阶段将每项对应的事务编号合并起来。统计事务编号数大于最小支持度的项，产生频繁一项集，并将该项及包含该项的事务序列进行存储。

传统的 Eclat 算法将事务以（ItemSet，TidSet）形式存储并操作，但在求候选项集支持度时，巨大的数据量会导致 TidSet 规模庞大，TidSet 的交集操作将消耗大量时间，影响算法效率。为解决上述问题，SPEclat 算法改变了数据的存储方式，将数据以(ItemSet，BitSet)的方式进行存储，这样在支持度计数时，可以将集合求交的运算改进为 BitSet 求与，避免了大数据量的运算。

第一阶段主要任务为获取频繁一项集，将事务中的项进行分隔得到(Tid,Items)键值对，并以事务项为键把包含该项的事务编号进行聚合，形成(ItemSet,BitSet)键值对，最后将支持度大于最小阈值的项集键值对进行筛选，得到频繁一项集，使用算法的伪代码如算法 1 所示。

算法 1 获取频繁一项集

```
For each transaction t(Tid,Items) in T
flatMap(line,t)
  Foreach item I in t
    out(I,Tid)
  ReduceByKey(I,Tid)
while(Item I in partition)
  BitSet+=Tid
filter( |BitSet| >=support)
  L1+=(I,BitSet)
end
```

存储结构的转换如图 1 所示，在转换后的存储结构中，r_{xy} 代表 T_y 中是否包含 I_x，若包含，r_{xy} 为 1，否则为 0。此时，如果需要计算某 k 项集在整个数据集中的出现次数，只需将存储数据中这 k 个项对应的 BitSet 求与，操作结果就是所有包含这个 k 项集的事务编号，计数即可得到其出现次数。

Tid	ItemSet
T_1	I_1, I_2, I_3, I_4, I_5 …… I_n
T_2	I_1, I_2, I_3, I_4, I_5 …… I_n
…	……
T_3	I_1, I_2, I_3, I_4, I_5 …… I_n
T_4	I_1, I_2, I_3, I_4, I_5 …… I_n

Item	TidSet(BitSet)					
I_1	r_{11}	r_{12}	r_{13}	r_{14}	…	r_{1n}
I_2	r_{21}	r_{22}	r_{23}	r_{24}	…	r_{2n}
I_3	r_{31}	r_{32}	r_{33}	r_{34}	…	r_{3n}
…					…	
I_n	r_{41}	r_{42}	r_{43}	r_{44}	…	r_{4n}

图 1 数据存储结构转换图

2.2 第二阶段

在这一阶段，主要实现对频繁项集按前缀进行划分。以频繁二项集的产生与划分为例，首先，使用频繁一项集两两相交，得到二项候选项。然后，使用对应的两个 BitSet 求与，并对新产生的 BitSet 计数，得到频繁二项集，将频繁二项集以(items，BitSet)的形式存储，并根据首项前缀分布到不同的计算节点。因为 BitSet 求与操作常数时间即可完成，故频繁项集产生的消耗是非常少的。数据划分的伪代码如算法二所示。

在各节点上使用 SPEclat 算法并行挖掘频繁集，需要迭代地以 map/reduce 任务来实现，其中 map()方法实现数据的划分，reduce 阶段实现改进后的 Eclat 算法。在 mapper 和 reducer 之间需要重写 partition 过程以确保属于同一分组的前缀所对应的列被划分到同一个计算节点。

算法 2 按前缀划分数据

```
Read L(2) from RDD
map(items,BitSet)
  get the profix p from items
  out(p, (items, BitSet))
  ReduceByKey(p, (items,BitSet))
    get all C_p from items
Eclat( C_p )
    out(frequent Itemset with profix p)
    end
```

2.3 第三阶段

在这一阶段，将在各个计算节点中针对具有相同前缀的项集，并行地以自底向上的形式进行迭代，用频繁 k 项集产生频繁(k+1)项集。即在每个节点中，对分配到该节点的项集进行自连接，产生候选 k 项集，之后进行支持度计数。

因为在上一步，数据已经被转换为(ItemSet，BitSet)形式，故求候选项集支持度时，可使用在同一节点的两个自连接的频繁项集的 BitSet 求与来对候选项集的进行计数。在求与之后，得到包含该候选项集的事务序列。该过程中，利用 BitSet 求与的快速运算代替集合之间求交运算，能节省大量时间。之后对生成的 BitSet 进行计数，判断计数值是否大于最小支持度，若大于，则将候选项集和对应 BitSet 组成的键值对，存入当中。算法 3 是第三阶段所采用算法的伪代码。

之后，对生成的频繁(k+1)项集进行再划分，迭代式地调用 SPEclat 算法挖掘频繁项集，直到没有更多的频繁项产生。

算法 3 生成频繁 k+1 项集

```
Read L(k) from RDD
for each li in L(k)
  for each lj in L(k)
    ItemSet=(li[1],li[2],li[3]…li[k-1],li[k],lj[k])
    BitSet = li.BitSet ∩ lj.BitSet
      if( |BitSet| >=support)
        L(k+1)+=(ItemSet,BitSet)
```

end

2.4 时间复杂度分析

需要对一些值进行假设，以数学公式的形式表示算法的时间复杂度。假设原数据集中事务个数为 T，计算节点数为 N，t 代表频繁一项集的个数。原算法的时间复杂度为 O(t^2+T/N×a^2)。而对原算法进行改进后的 SPEclat 算法的时间复杂度为 。

3 相关实验与分析

实验室的分布式计算平台由九台计算机搭建而成，其中主节点配置为 i7 3770 处理器，4GB 内存，1TB 硬盘。计算节点配置为奔腾 e2140 处理器，1GB 内存，250GB 硬盘,Spark 集群的环境参数如下所示：Ubuntu:14.04;Hadoop version:2.4.0;Spark version:1.6.1。

为了验证 SPEclat 算法的正确性，在 T10I4D100K、Pumsb_star 和 chess 数据集上与 MREclat 的处理结果进行了对比，实验表明两种算法得到的结果一致。

3.1 加速比测试

为了验证 SPEclat 算法是否具有可扩展性以及处理大数据的能力，拟人工生成大小为 10 GB、20 GB、30 GB，项目数为 80 的三个不同大小的数据集，用于下一步实验。

加速比指同一任务在单处理器和多处理器系统中运行消耗时间的比率，是一种用来衡量并行化性能的一个有效指标。因此为了考察算法的并行化性能，应用以上数据进行算法的加速比性能实验。实验对 3 个数据集分别在 4，6，8 个节点上运行算法，实验结果如图 2 所示。为了对比明显，加入了线性加速比结果。

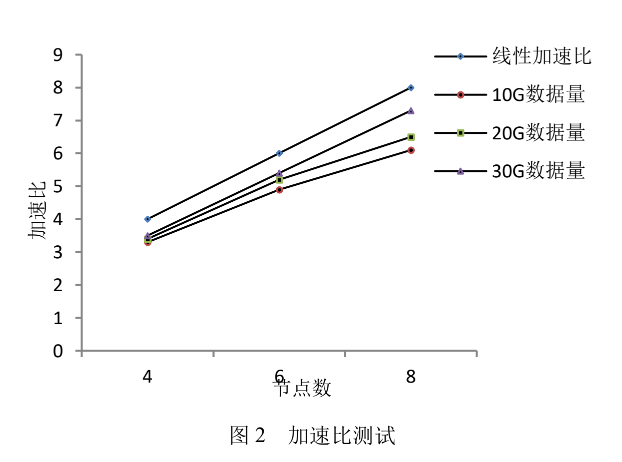


图 2 加速比测试

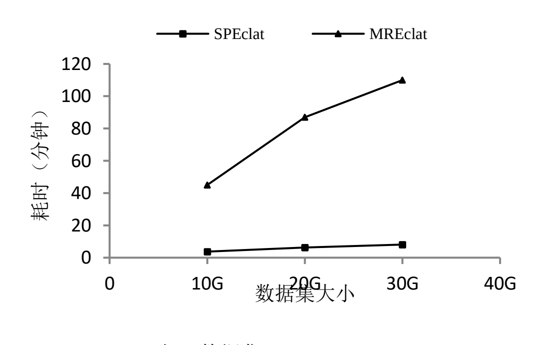
从图 3 可以看出，对于同一数据集，该算法运行的加速比会随着节点数的增加而增加，由于节点数的增加使得节点间的通信、数据传递所消耗的时间逐渐增加，故增加趋势接近线性，但是与线性加速比有一定差距，而且随着节点数的增加差距也越大。对比不同数据集的加速比，当数据量越大时，加速比会越接近线性加速比，原因在于：面对数据量更大的数据集，数据计算所消耗的时间占总耗时的比例相比于小的数据集要更多。

实验表明，SPEclat 具有一定的可扩展性。

3.2 数据的可扩展性测试

这一节通过比较 SPEclat 算法与 MREclat 算法在不同数据集上的运算效率，验证 SPEclat 的数据可扩展性。SPEclat 是 Eclat 算法在 Spark 上的首次实现，而 MREclat 是 Eclat 算法基于 MapReduce 的经典实现，其运行效率较其他同类算法更为优越，所以用这两个算法进行对比，以验证 SPEclat 的数据可扩展性。

首先使用上一实验中人工生成的数据集，对 SPEclat 算法与 MREclat 算法进行了对比实验，以评估算法的运行效率。其中 x 轴代表人工数据的数据量，y 轴则表示程序执行所用的时间，支持度为 5%，节点数设置为八个。实验结果如图 3 所示。



人工数据集: min_sup = 5%

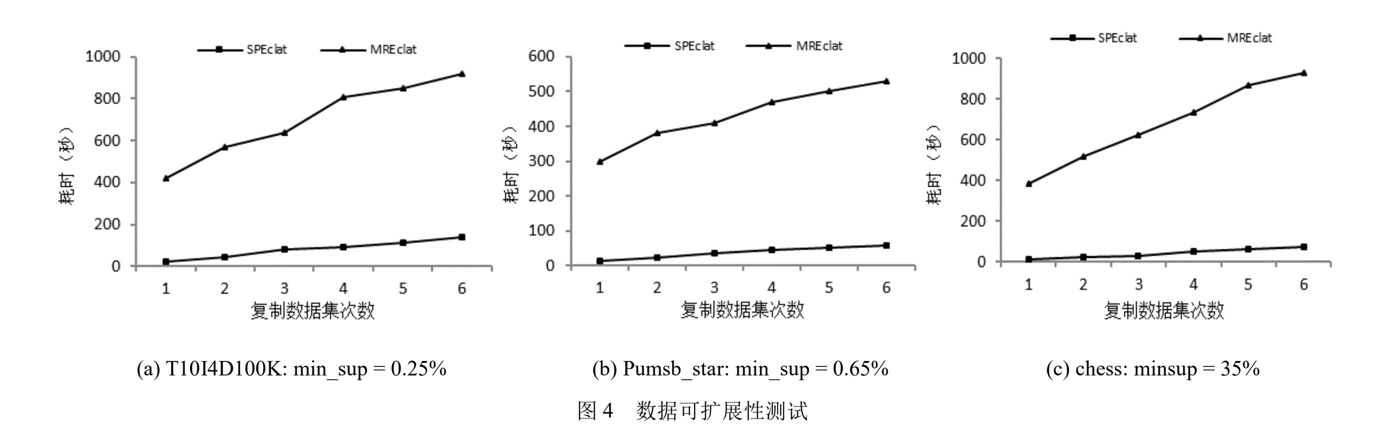
图 3 人工数据测试实验

实验的统计结果表明，SPEclat 算法较 MREclat 算法而言，在性能上取得了显著地提高，并且随着数据量的增长，性能的提升不断增加，相较于 MREclat 算法，且 SPEclat 算法的性能具有明显优势。

为了进一步验证算法的数据可扩展性，对 SPEclat 算法与 MREclat 算法分别进行了多组对比实验。使用公共测试数据集 T10I4D100K、Pumsb_star 和 chess 以保证实验的客观性。为了确保实验结果的正确性，图中的值为算法运行 10 次的平均值。

在测量算法可扩展性的过程中，将节点数调整为八个，并分别将测试数据复制为原数据集大小的 2、3、4、5、6 倍，通过多次实验测量两个并行算法的数据可扩展性。实验结果如图 4(a)~(c)所示。其中 x 轴代表副本数据的复制次数，y 轴则表示程序执行所花费的时间。从图中可以看到，对于不断增长的数据量，MREclat 算法执行所消耗的时间近似于直线式增长，而 SPEclat 算法的运算时间增长较为缓慢，基本上保持水平。同时 SPEclat 算法所消耗的时间远少于 MREclat，完成计算任务的速率是 MREclat 算法速率的 20 倍左右。

其原因是 SPEclat 算法采用基于内存的计算方式，可将数据集在内存中快速地进行多次迭代，省去大量 I/O 操作。同时，在对候选项集进行支持度计数时，通过对数据存储结构的改变，实现了利用 BitSet 求与的快速运算代替集合之间求交运算，优化了支持度计数的计算方式，提升了运算效率，且相较于 MREclat 算法，SPEclat 算法性能的提升会随着数据量的增加而变得愈发明显。

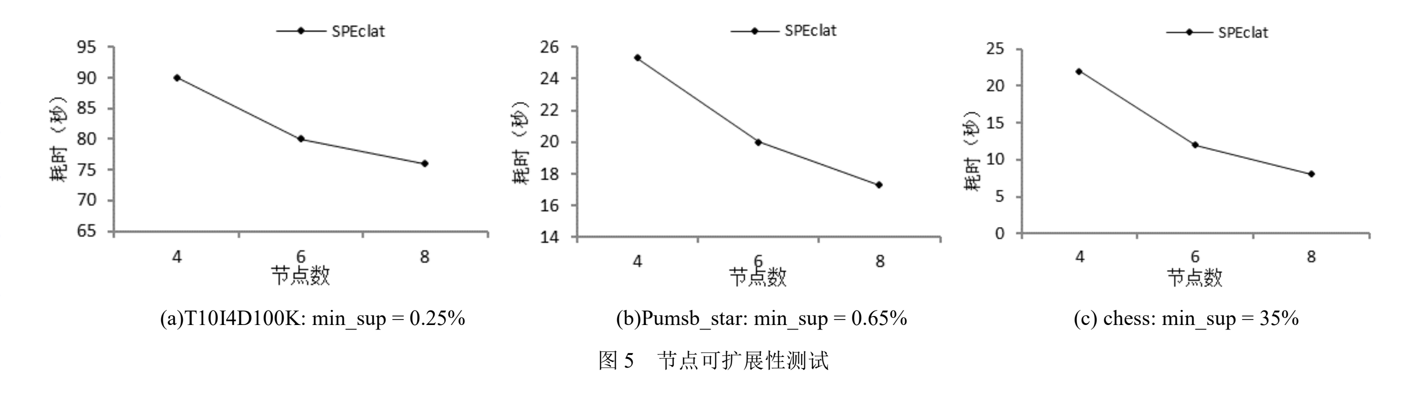


(a) T10I4D100K: min_sup = 0.25%　(b) Pumsb_star: min_sup = 0.65%　(c) chess: minsup = 35%

图 4　数据可扩展性测试

3.3　节点的可扩展性测试

在验证 SPEclat 算法的节点可扩展性时，分别使用公共测试数据集 T10I4D100K、Pumsb_star 和 chess 进行实验，并调整集群节点数为 4、6、8 个，保持数据集的大小不变，分别记录 SPEclat 算法执行所消耗的时间，结果如图 5(a)、(b)、(c)所示，其中 X 轴代表集群中节点的个数，Y 轴代表所花费的时间。分析三实验图可知，随着集群中节点数的增加，SPEclat 算法的耗时呈近似线性依次递减，结果表明，该算法具有良好的节点可扩展性。



(a)T10I4D100K: min_sup = 0.25%　(b)Pumsb_star: min_sup = 0.65%　(c) chess: min_sup = 35%

图 5　节点可扩展性测试

4　结束语

本文提出了基于 Spark 的 Eclat 频繁项集挖掘算法 SPEclat。针对信息化时代产生的海量数据，本文将原有串行 Eclat 算法进行了改进，并依托 Spark 平台将其实现并行化，充分利用 Spark 平台基于内存、利于迭代式计算的特性，完成了算法在挖掘海量数据时性能上的突破，解决了其并行化所遇到的，诸如数据的划分，生成频繁项集时产生大量的交叉计算等问题。最终利用人造数据以及公共数据集加以实验，证明针对该算法的改进是行之有效的，且随着数据量的不断增长，算法的性能会有更加显著的提高。

参考文献：

[1] HanJiawei, Kamber M, Pei Jian, 等. 数据挖掘：概念与技术 [M]. 北京: 机械工业出版社, 2012.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases [C]// Proc of International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc. , 1994: 487-499.

[3] Zaki M J, Parthasarathy S, Ogihara M, et al. Parallel algorithms for discovery of association rules [J]. Data Mining and Knowledge Discovery, 1997, 1 (4): 343-373.

[4] Zaki M J. Scalable algorithms for association mining [J]. IEEE Trans on Knowledge & Data Engineering, 2000, 12 (3): 372-390.

[5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [C]// Proc of the 6th Conference on Symposium on Opearting Systems Design & Implementation. Berkeley: USENIX Association, 2004: 107-113.

[6] Deng L, Lou Y. Improvement and research of FP-growth algorithm based on distributed Spark [C]// Proc of International Conference on Cloud Computing and Big Data. Washington DC: IEEE Computer Society, 2015: 105-108.

[7] Yang S, Xu G, Wang Z, et al. The parallel improved Apriori algorithm research based on Spark [C]// Proc of the 9th International Conference on Frontier of Computer Science and Technology. 2015: 354-359.

[8] Wang S, Wu P, Liu T, et al. P-WLPA algorithm research on parallel framework Spark [C]// Proc of Information Technology and Artificial Intelligence Conference. 2014: 437-441.

[9] 李伟卫，赵航，张阳，等. 基于 MapReduce 的海量数据挖掘技术研究 [J]. 计算机工程与应用, 2013, 49 (20): 112-117.

[10] 章志刚，吉根林，唐梦梦. 并行挖掘频繁项目集新算法——MREclat [J].

计算机应用, 2014, 34 (8): 2175-2178.

[11] 曹博, 倪建成, 李淋淋, 等. 基于 Spark 的并行频繁模式挖掘算法 [J]. 计算机工程与应用, 2016, 52 (20): 86-91.

[12] Qiu H, Gu R, Yuan C, et al. YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark [C]// Proc of IEEE International Parallel & Distributed Processing Symposium Workshops. Washington DC: IEEE Computer Society, 2014: 1664-1671.

[13] 陈明洁. 分布式频繁项集挖掘算法 [J]. 计算机应用与软件, 2015 (10): 63-66.

[14] 赵焱德. 基于 SPARK 的海量数据频繁模式挖掘算法研究 [D]. 哈尔滨: 哈尔滨工业大学, 2016.

[15] 郭进伟, 皮建勇. 基于 MapReduce 的 SON 算法实现 [J]. 计算机应用, 2014 (a01): 100-102.

[16] 马可, 李玲娟, 孙杜靖. 分布式并行化数据流频繁模式挖掘算法 [J]. 计算机技术与发展, 2016 (7): 75-79.

[17] 唐颖峰, 陈世平. 一种基于后缀项表的并行闭频繁项集挖掘算法 [J]. 计算机应用研究, 2014, 31 (2): 373-377.